快速变换

2019年8月27日 15:43

-
- FFT

1) n次单位根：满足$\omega^n=1$的复数ω
2) $\omega_n^k = e^{2k\pi i / n}$，k取[0,n)
3) n个ω均匀地分布在复平面上半径为1的圆上
4) $\omega_n^n = e^{2\pi i} = 1$
5) 欧拉公式：$e^{ix} = \cos x + i \sin x$

1. 消去引理：$\omega_n^k = \omega_{dn}^{dk}$（dnk取正整数）
   1) Cor：n为正偶数时，$\omega_n^{n/2} = -1 = \omega_2$
2. 折半引理：n为正偶数时，$(\omega_n^{k+n/2})^2 = \omega_n^{2k+n} = \omega_n^{2k}\omega_n^n = \omega_n^{2k} = (\omega_n^k)^2$
   1) Cor：n为正偶数时，n个n次单位复数根的平方的集合=n/2个n/2次单位复数根的平方的集合
3. 求和引理：$\Sigma(\omega_n^k)^j=0$（n正整数，k非负整数且不是n的倍数，j取[0,n)）
   1) 可由几何级数（等比数列求和公式）推出
4. 多项式卷积c=a*b
   1) 其中abc都是系数向量，若a和b都是n次，则c将是2n次多项式
   2) $c_j=\Sigma a_k b_{j-k}$(k取[0,j))
5. 多项式点值表示法：将n个不同的x代入多项式，得到n个y，用这n个(x,y)对来表示
   1) Discrete Fourier Transform离散傅立叶变换：把系数表达式映射到点值表达
   2) Inverse DFT逆变换IDFT：再映射回去，利用范德蒙德矩阵的逆，不难发现改一个符号就可以实现了
   3) Fast FT快速傅变FFT：利用n个n次单位根做DFT，每次迭代可折半乘法次数
6. 蝴蝶定理
   1) 长度恰为2的整数次幂的序列，按奇偶性进行递归（下标从0开始，偶数下标递归进左子树），得到的叶结点序下标与原下标恰在二进制下是相反的01串
   2) 设01串长ln，可以将i的逆串存进r[i]= (r[i>>1] >>1) | ((i&1)<< ln-1)
   3) 于是直接用r[i]既可求得原各i对应的叶i，先用j枚举结点长度，再用k遍历j上层各结点，再用l遍历j层各数值即可代替递归
7. 存储空间优化+算法复杂度常数优化
   1) 一般序列A和B都是不需要再利用的，所以存储复数乘积可以直接存进A或B
   2) 易知$(a+bi)^2 = (a^2-b^2) + 2abi$，所以将右操作数序列B读进左序列A的虚部，再输出平方后A的虚部的二分之一即可（稍微有一点点担心这个精度问题？）
8. 碎碎念
   1) 要用fft的题多半也要快读，而fft必须用double型存数值，好在int型作为右值赋给double左值变量是可以的，只是scanf不能直接赋值%d给double变量
   2) 快写的优化好像不是特别明显

3) printf的%.f可能会出现-0之类的情况，要用+1e-8玄学解决，所以还是手动int(浮点数+0.5)再用整数输出吧

◆

◆ FFT模板

```
typedef double db;

const int MN = 3e6 + 5;
const db pi = acos(-1);

struct CP{
     db x,y;      //实部，虚部
     CP (db x=0, db y=0): x(x), y(y) {}
     CP operator+(CP &t){ return CP(x+t.x, y+t.y); }
     CP operator-(CP &t){ return CP(x-t.x, y-t.y); }
     CP operator*(CP &t){ return CP(x*t.x - y*t.y, x*t.y + y*t.x); }
}a[MN],b[MN],c[MN];

int read(){
     int k=0, f=1;
     char c= getchar();
     while(!isdigit(c)){ if(c=='-') f= -1; c= getchar(); }
     while(isdigit(c)) k= k*10 + c-48 , c= getchar();
     return k*f;
}
void write(int x){
     if(x<0){ putchar('-'); x=~(x-1); }
     int s[20],top=0;
     while(x){ s[++top]=x%10; x/=10; }
     if(!top) s[++top]=0;
     while(top) putchar(s[top--]+'0');
}

int N,M,n,ln;//左多项式次数，右多项式次数，fft次数及其位数
int R[MN];   //01串逆转后对应的下标
void init(){
     N= read(); M= read();
     for__(i,0,N) a[i].x= read();
     for__(i,0,M) b[i].x= read();
     n=1;
     while(n<=N+M) n<<=1, ++ln;
     for_(i,1,n) R[i]= (R[i>>1] >>1) | ((i&1)<< ln-1);
}
void fft(CP c[], int f=1){//f取-1时是逆变换
     for_(i,0,n) if(i<R[i]) swap(c[i], c[R[i]]);
     for(int j=1; j<n; j<<=1){
          CP wn(cos(pi/j), f*sin(pi/j));
          for(int k=0; k<n; k+= (j<<1)){
               CP t(1, 0);
               for_(l,0,j){
                    CP cl= c[k+l], cr= t* c[j+ k+l];
                    c[k+l]= cl+cr;
                    c[j+ k+l]= cl-cr;
                    t= t*wn;
               }
          }
     }
}

     init();
     fft(a), fft(b);
     for__(i,0,n) c[i]= a[i]*b[i];
     fft(c,-1);
     for__(i,0,N+M) write(int(c[i].x/n + 0.5)), putchar(' ');
```

-
- NTT

1. 原根g：当g^i mod p结果两两不相同，共能取到phi(p)种时，称g为p的原根
   1) 质数p的原根g：g^i mod p有p-1种取值（2<= g <=p-1；1<= i <=p-1）
   2) 例：3是质数998244353的原根
2. 快速数论变换Number Theory Transform
   1) 利用原根g代替复数单位根ω实现的快速变换
   2) fft中利用(cos(pi/j),sin(pi/j))实现了单位根e^ i*pi/j，将它换成g^ (p-1)/(2j)
   3) 最后除以n的时候换成乘以n的逆元
   4) 注意乘法结果可能爆int，要用long long存乘法结果，以及注意及时取余P
3. 分治卷积：形如f[n]=∑_{i=0}^{n-1}f[i]g[n-i]
   1) 因为求f[n]会用到f[0]~f[n-1]，所以不能直接用NTT
   2) 尝试用cdq分治区间[l,r]：
      i. f[x] = ∑_{i=l}^{x-1}f[i]g[x-i] = ∑_{i=0}^{x-1-l}f[i-l]g[x-i-l]
      ii. 令a[i]=f[l+i]，b[i]=g[i+1]，N=x-1-l
      iii. 则f[x] = ∑_{i=0}^{N}a[i]b[N-i]，是常规的FFT卷积，按上式的令法，在[l,mid]分治完后，将[l,mid]的f存进a，[0,r-1)的g存进b，对ab进行NTT，即可求出[l,mid]的f对[mid+1,r]的贡献，再分治[mid+1,r]
      iv. 由于分治本身是O（nlogn）的算法，每次分治中还要算一次卷积，算法复杂度至少是O（n log^2 n）

```
const int P = 998244353;
const int G = 3;
const int MN = 2e5 + 5;

inline int read(){
	int k=0, f=1;
	char c=getchar();
	while(!isdigit(c)){ if(c=='-') f=-1; c=getchar(); }
	while(isdigit(c)) k=k*10+c-48, c=getchar();
	return k*f;}

int qpow(ll a, int b){
	ll ans=1ll;
	for(;b;b>>=1){
		if(b&1)
			ans= ans*a %P;
		a= a*a %P;}
	return ans;}

inline int inv(ll value){ return qpow(value, P-2); }

namespace NTT{
	int R[MN];	//01串逆转后对应的下标

	void calcR(int len){		//计算len次多项式的R数组
		int loglen = R[0] = 0;
		while(1<<loglen < len) ++loglen;
		for_(i,1,len) R[i] = (R[i>>1] >>1) | ((i&1)<< loglen-1);
	}
	void ntt(int *a, int n, int f=1){		//f取-1时是逆变换
		for_(i,0,n) if(R[i] < i) swap(a[i],a[R[i]]);
		int baseW = qpow(G, (P-1) / n);
		if(f==-1) baseW = inv(baseW);
```

```
        for(int len = 2; len <= n; len <<= 1){
            int mid = len >>1;
            int wn = qpow(baseW, n / len);
            for(int *pos = a; pos != a+n; pos += len){
                int w = 1;
                for_(i,0,mid){
                    int x = pos[i], y = (ll)pos[mid + i] * w % P;
                    pos[i] = ((ll)x + y) % P;
                    pos[mid + i] = ((ll)x - y + P) % P;
                    w = (ll)w * wn % P;
                }
            }
        }
    }
    void mult(int *a, int *b, int len){     //len次多项式乘法，答案存进a
        calcR(len);
        ntt(a,len);
        ntt(b,len);
        for_(i,0,len) a[i] = (ll)a[i] * b[i] % P;
        ntt(a,len,-1);
        int x = inv(len);
        for_(i,0,len) a[i] = (ll)a[i] * x % P;
    }
}
using namespace NTT;

int f[MN],g[MN],A[MN],B[MN];      //输出输入及其副本
int n; //多项式长度

void cdq(int l, int r){
    if(l==r) return;
    int mid = l+r >> 1, len = 1;
    while(len < r-l) len <<= 1;
    cdq(l,mid);
    for_(i,0,len) A[i] = 0, B[i] = 0;
    for__(i,0,mid-l) A[i] = f[l+i];  //l开始的当前区间的左半f
    for_(i,0,r-l) B[i] = g[i+1];     //1开始的当前区间长度的g
    mult(A,B,len);       //卷积求出左半端的贡献
    for__(i,mid+1,r) f[i] = (f[i] + A[i-l-1]) % P;
    cdq(mid+1,r);
}
    n = read();
    for_(i,1,n) g[i] = read();
    f[0] = 1;     //题目给的初值
    cdq(0,n-1);
    for_(i,0,n) printf("%d ",f[i]);
```

4. 多项式求逆：求F使得F(x)G(x) ≡ 1（除了模P外还要模x^n：忽略n次及更高次项）
    1) 设G[0]为G的0次项，先求得G[0]在P的逆
    2) 尝试倍增求解：设f是前n项的逆，F是前2n项的逆，则模x^n时，G(F-f)=0，G不为0时F-f的前n项相减为0，则平方后在模x^2n时，F方+f方-2Ff=0
    3) 两边同乘G，得到F+Gf方-2f=0，于是得到了递增求G前2^n项逆F的方法
    4) 时间复杂度约6倍DFT，依旧为O（nlogn）（板子莫名原因不能做分治）

```
const int P = 998244353;
const int G = 3;
const int MN = 4e5 + 5;

inline int read(){
    int k=0, f=1;
    char c=getchar();
```

```
	while(!isdigit(c)){ if(c=='-') f=-1; c=getchar(); }
	while(isdigit(c)) k=k*10+c-48, c=getchar();
	return k*f;}

int qpow(ll a, int b){
	ll ans=1ll;
	for(;b;b>>=1){
		if(b&1)
			ans= ans*a %P;
		a= a*a %P;}
	return ans;}

inline int inv(ll value){ return qpow(value, P-2); }

	int R[MN];	//01串逆转后对应的下标

	void calcR(int len){	//计算len次多项式的R数组
		int loglen = R[0] = 0;
		while(1<<loglen < len) ++loglen;
		for_(i,1,len) R[i] = (R[i>>1] >>1) | ((i&1)<< loglen-1);
	}


	void ntt(int *a, int n, int f=1){	//f取-1时是逆变换
		for_(i,0,n) if(R[i] < i) swap(a[i],a[R[i]]);
		int baseW = qpow(G, (P-1) / n);
		for(int len = 2; len <= n; len <<= 1){
			int mid = len >>1;
			int wn = qpow(baseW, n / len);
			for(int *pos = a; pos != a+n; pos += len){
				int w = 1;
				for_(i,0,mid){
					int x = pos[i], y = (ll)pos[mid + i] * w % P;
					pos[i] = ((ll)x + y) % P;
					pos[mid + i] = ((ll)x - y + P) % P;
					w = (ll)w * wn % P;
				}
			}
		}
		if(f==-1){
			int miv = inv( ll(n) );
			reverse(a+1,a+n);
			for_(i,0,n) a[i] = ll(a[i]) * miv % P;
		}
	}
//void ntt(int *a,int n,int x=1) {
//	#define RI register int
//	#define mod P
//	#define ksm qpow
//	for(RI i=0;i<n;++i) if(i<R[i]) swap(a[i],a[R[i]]);
//	for(RI i=1;i<n;i<<=1) {
//		RI gn=ksm(G,(mod-1)/(i<<1));
//		for(RI j=0;j<n;j+=(i<<1)) {
//			RI t1,t2,g=1;
//			for(RI k=0;k<i;++k,g=1LL*g*gn%mod) {
//				t1=a[j+k],t2=1LL*g*a[j+k+i]%mod;
//				a[j+k]=(t1+t2)%mod,a[j+k+i]=(t1-t2+mod)%mod;
//			}
//		}
//	}
//	if(x==1) return;
//	int ny=ksm(n,mod-2); reverse(a+1,a+n);
//	for(RI i=0;i<n;++i) a[i]=1LL*a[i]*ny%mod;
//}

int f[MN],g[MN],A[MN],B[MN];	//输出输入及其副本

int n; //多项式长度
```

```
void inv(int *a, int *b, int len){      //len次多项式求逆，答案存进b

    if(len==1) return b[0] = inv(ll(a[0])), void();
    inv(a, b, len+1 >>1);
    int LEN = 1; //扩展到2的整数次幂的len
    while(LEN < (len<<1)) LEN <<= 1;
    calcR(LEN);
    for_(i,0,len) A[i] = a[i];
    for_(i,len,LEN) A[i] = 0;
    ntt(A,LEN), ntt(b,LEN);
    for_(i,0,LEN) b[i] = ll(2 - ll(A[i]) * b[i] % P + P) % P * b[i] % P;
    ntt(b,LEN,-1);
    for_(i,len,LEN) b[i] = 0;
}

    n = read();
    for_(i,0,n) g[i] = read();
    inv(g,f,n);
    for_(i,0,n) printf("%d ",f[i]);
```